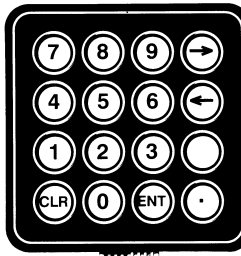
 Friedrich-Ebert-Schule Esslingen	Projekt: PC-Diagnose-Display	Name:
	6.2.1	Bestellnummern und Bestückung



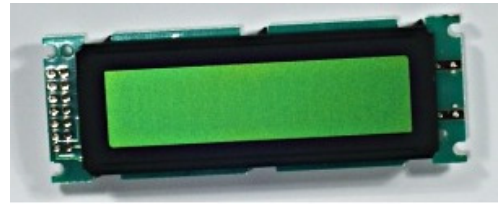
Matrix-Folientastaturen

z.B. RS-electronics
 4x1 Matrix Bst.-Nr. 130-381 13,35€
 4x3 Matrix Bst.-Nr. 130-397 15,35€
 4x4 Matrix Bst.-Nr. 130-404 15,35€

z.B. Conrad
 4x3 Matrix 6,25€

Wichtig: Matrix-Folientastaturen erkennt man an der Anzahl der Anschlüsse!

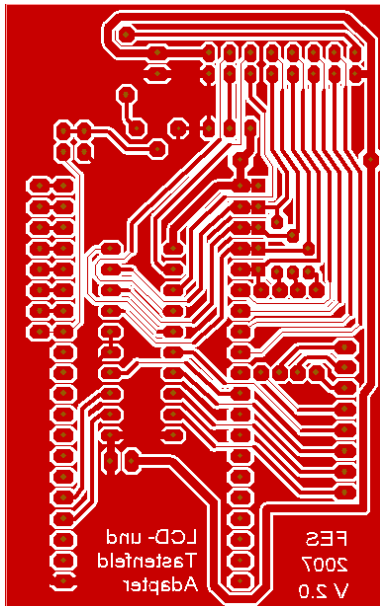
Beispiel: 4x4-Matrix → 8 Anschlüsse
 4x3-Matrix → 7 Anschlüsse
 4x1-Matrix → 5 Anschlüsse



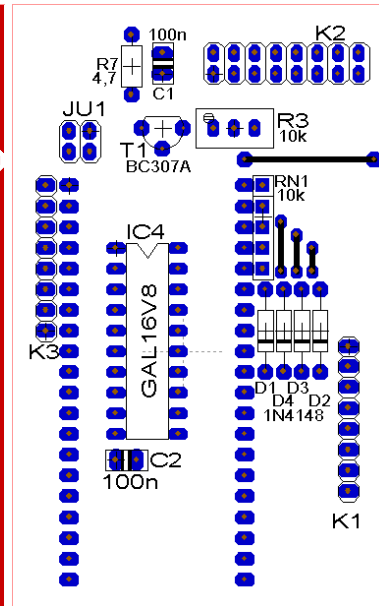
LC-Display

z.B. Reichelt
 4x16 Zeichen mit Backlight Bst.-Nr. LCD 164A LED 16,95€
 4x16 Zeichen ohne Backlight Bst.-Nr. LCD 164A 16,95€
 4x20 Zeichen Bst.-Nr. LCD 204B LED 17,50€

Layout (gespiegelt)



Bestückung

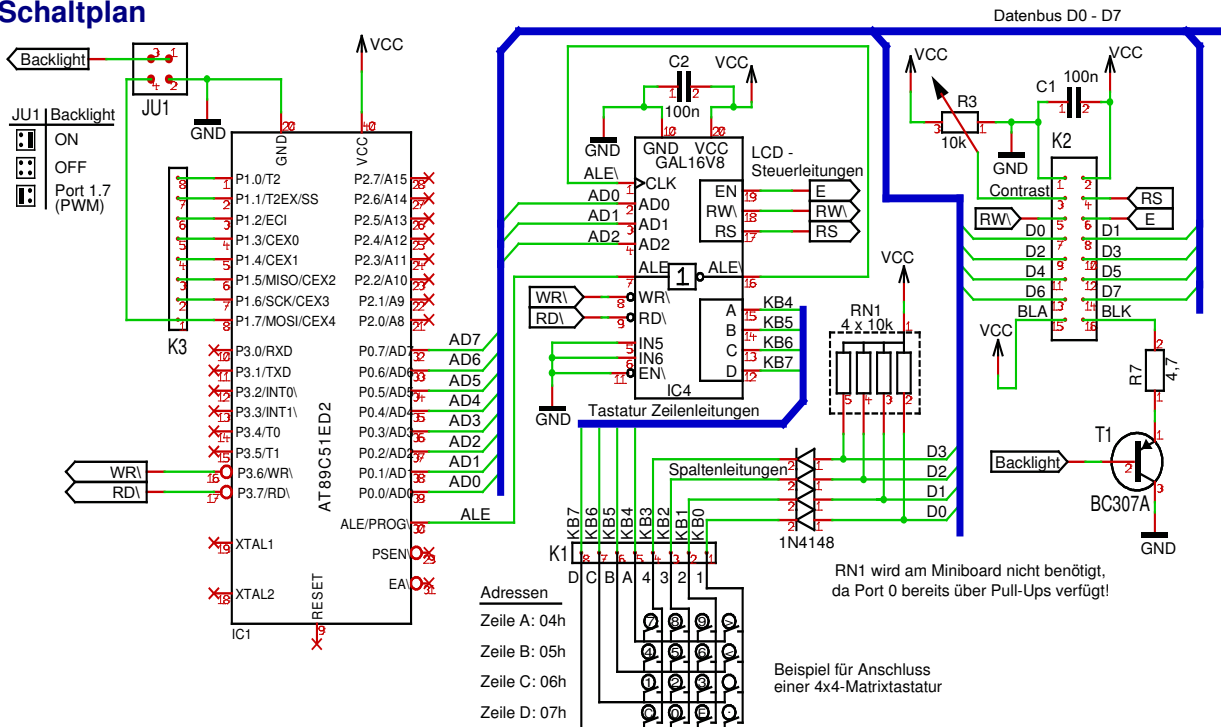



Stückliste

z.B. Reichelt

Name	Anz	Wert	Best.-Nr.	Preis
D1..D4	4	1N4148	1N4148	0,08€
R7	1	4,7	1/4W 4,7	0,10€
RN1	1	10k	SL 5-4 10k	0,08€
R3	1	10k	64W-10k	0,64€
C1,C2	2	100n	Z5U-2,5 100N	0,12€
T1	1	BC307A od. BC558C	BC 558C	0,04€
IC4	1	GAL16V8	GAL16V8-25LP	0,82€
	1	IC-Sockel	GS20P	0,23
K1,K3	2	1x20	SL1X40G 2,54	0,54€
K2	1	2x2		
JU1	1	2x2		
K2	1	2x8	WSL 16G	0,07€
Buchse	1	2x8	PFL 16	0,09€
Flachbandkabel	1	16 polig	AWG28-16G 3M	2,75€

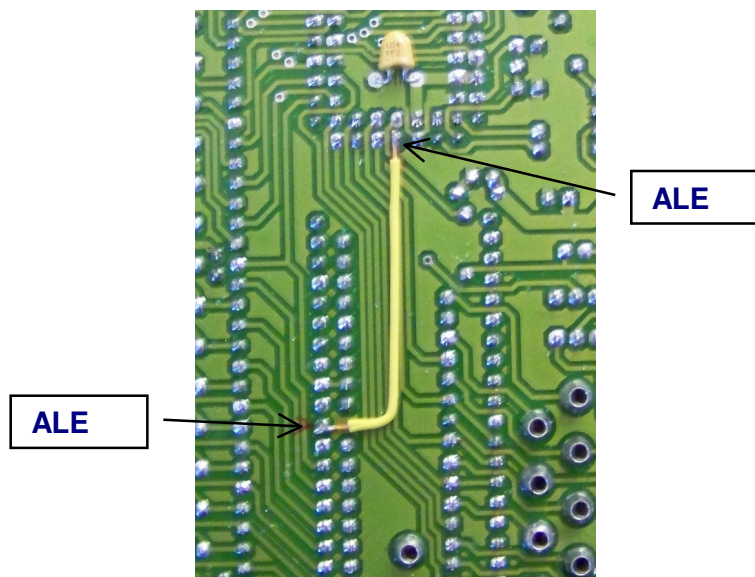
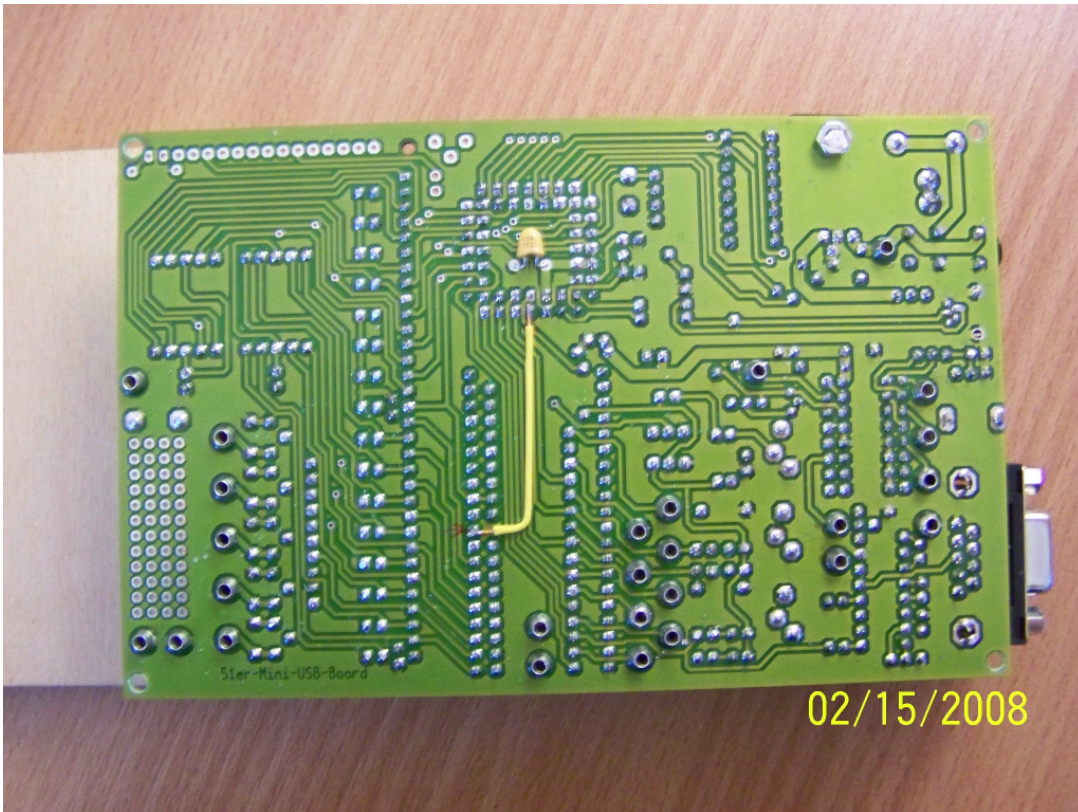
Schaltplan




 Friedrich-Ebert-Schule Esslingen	Projekt: PC-Diagnose-Display	Name:
6.2.2	ALE-Signal auf den neuen USB-Miniboards!	Datum:

Auf den neuen USB-Miniboards sind die Signale in der Reihenfolge der Belegung eines „normalen“ 40-poligen DIP-Controllers auf zwei 20-polige Buchsenleisten geführt. Somit können alle Zusatzplatinen, also auch die Display-Platine weitergenutzt werden.

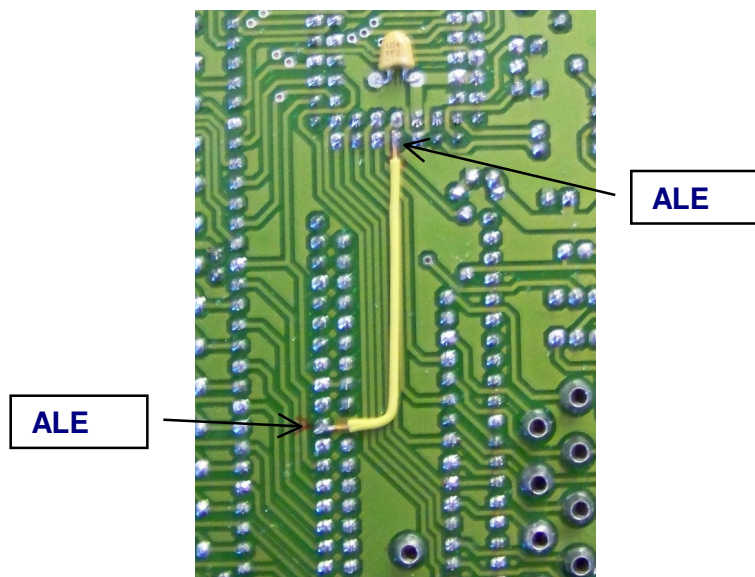
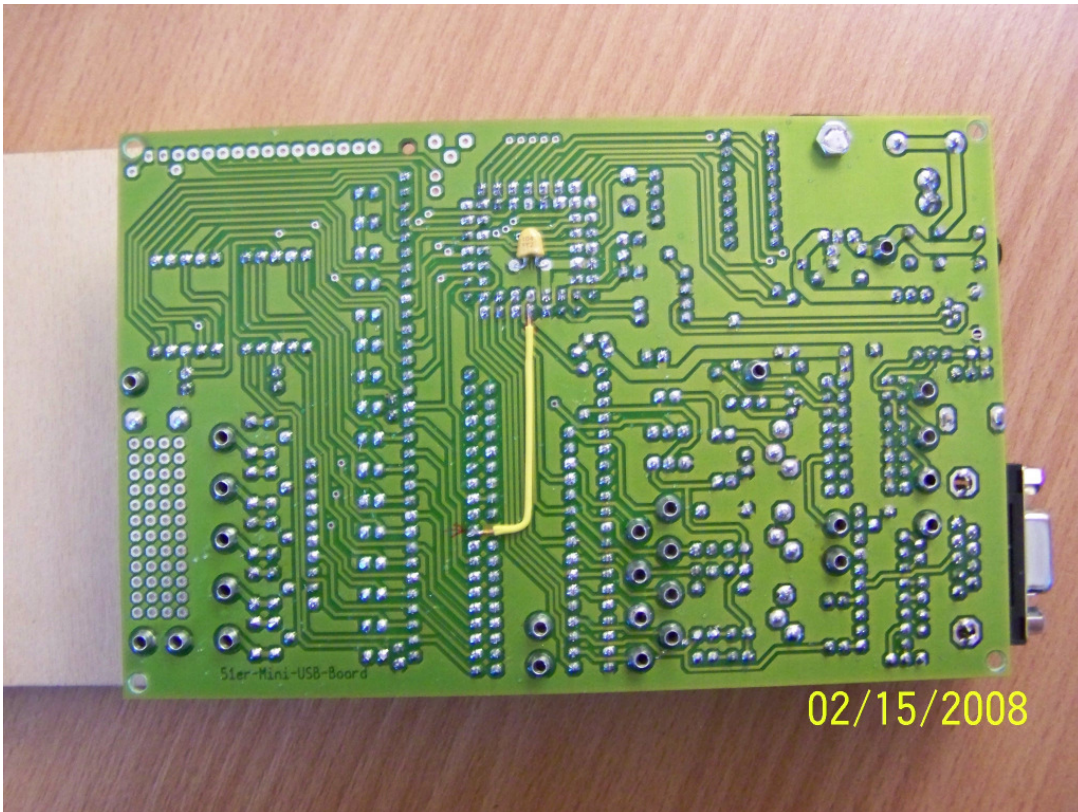
Leider fehlt als einziges das ALE-Signal, dass aber für die Funktion der Display-Platine unbedingt nötig ist. Das Foto zeigt wie mit einer Drahtbrücke das ALE-Problem gelöst werden kann. Die ALE Pins sind leicht zu identifizieren. Sie sind als einzige Pins nicht angeschlossen!




 Friedrich-Ebert-Schule Esslingen	Projekt: PC-Diagnose-Display	Name:
6.2.2	ALE-Signal auf den neuen USB-Miniboards!	Datum:

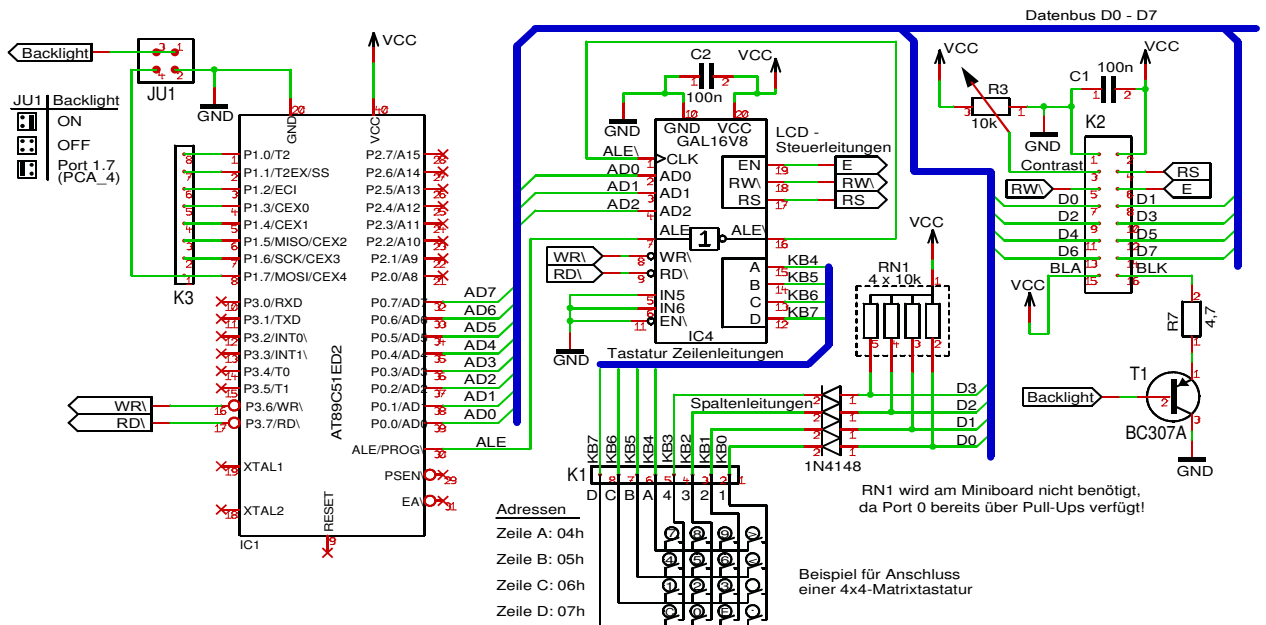
Auf den neuen USB-Miniboards sind die Signale in der Reihenfolge der Belegung eines „normalen“ 40-poligen DIP-Controllers auf zwei 20-polige Buchsenleisten geführt. Somit können alle Zusatzplatinen, also auch die Display-Platine weitergenutzt werden.

Leider fehlt als einziges das ALE-Signal, dass aber für die Funktion der Display-Platine unbedingt nötig ist. Das Foto zeigt wie mit einer Drahtbrücke das ALE-Problem gelöst werden kann. Die ALE Pins sind leicht zu identifizieren. Sie sind als einzige Pins nicht angeschlossen!



 Friedrich-Ebert-Schule Esslingen	Projekt: PC-Diagnose-Display	Name:
	6.2.2.1.1	GAL-Programmierung 1

Der GAL auf der Projektplatte erzeugt die Steuersignale für das LC-Display und für eine Matrixtastatur. Die Signale werden aus dem gemultiplexten Adress- und Datenbus (Port 0) des 8051-Controllers gewonnen. Der Mikrocontroller kann dann mit Schreib- und Lesebefehlen (movx) auf die entsprechenden externen Adressen zugreifen.



Arbeitsauftrag:

Programmiert die Steuersignale für LC-Display und Tastatur. Erstellt ein Projekt für den verwendeten GAL-Baustein (GAL-Aufdruck beachten) in DesignExpert und fügt die auf der folgenden Seite abgebildete ABEL-Datei ins Projekt ein. Bei ... muss fehlender Text eingefügt werden.

1. Programmiert das Signal **ALE/** als **equation**
 Die Steuersignale RS, RW, D, C, B, A müssen mit fallender Flanke von ALE **gespeichert** werden. Da die internen D-FFs des GAL auf steigende Flanke triggern, muss ALE zunächst invertiert werden. Anschließend wird ALE/ auf den CLK-Pin des GAL gelegt.
2. Programmiert die Signale **RS, RW, D, C, B, A** als **truth_table**
 Da nur 3 Adressbits benötigt werden, ergeben sich folgende Adresszuordnungen:
 Hinweis: Denke daran, dass diese Signale gespeichert sein müssen!!

Adressen			LCD		Tastatur				
hex	A2	A1	RS	RW	D	C	B	A	
00h	0	0	0	0	1	1	1	1	Kommando schreiben Busy-Flag lesen Daten schreiben Daten lesen } LC-Display
01h	0	0	1	1	1	1	1		
02h	0	1	1	0	1	1	1		
03h	0	1	1	1	1	1	1		
04h	1	0	X	X	1	1	1	0	Zeile A Zeile B Zeile C Zeile D } Tastatur
05h	1	0	X	X	1	1	0	1	
06h	1	1	X	X	1	0	1	1	
07h	1	1	X	X	0	1	1	1	

3. Programmiert das Steuersignal **E** als **equation**
E darf nur bei Schreibzugriffen (WR/ = 0) auf die LCD-Adressen 00h und 01h, sowie bei Lesezugriffen (RD/ = 0) auf die Adressen 01h und 03h H-Pegel haben.

```

MODULE LCD_Tastatur_decoder
@dcset
declarations
    CLK_IN      pin 1;
    ALE         pin 7;
    ALE_not     pin 16      istype'buffer,com';

    AD2,AD1,AD0 pin 4,3,2;  // gemultiplexerter Adress-/Datenbus (P0.2..0.0)
    !WR,!RD     pin 8,9;   // Schreib-/Lese-Leitungen vom µC (Nullaktiv!!)

    E          pin 19      istype'buffer,com';  // LCD Freigabesignal
    RS,RW      pin 17,18   istype'buffer,reg';  // LCD Steuersignale

    A,B,C,D    pin 15,14,13,12 istype'buffer,reg';  // Tastatur-Zeilen-Signale

    LCD        = [RS, RW];
    Keyboard   = [A,B,C,D];

equations
    ...

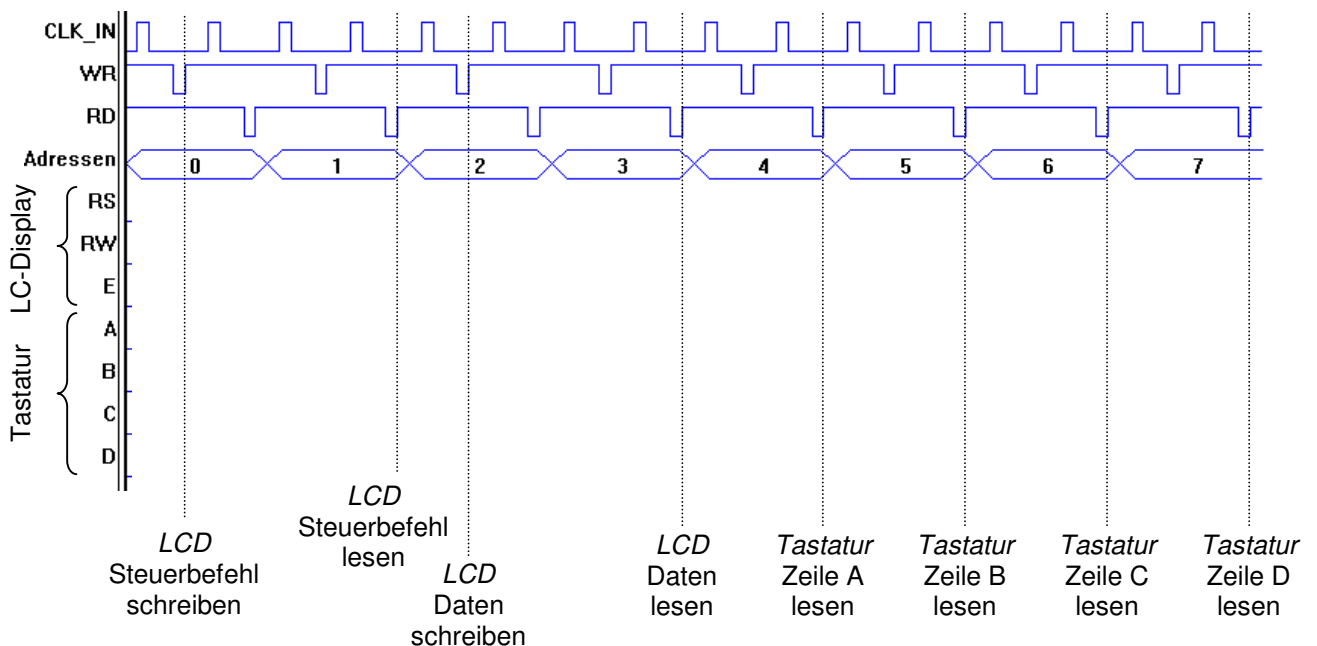
truth_table    // LCD- und Keyboard-Signale
    ...

test_vectors
    ([CLK_IN,!WR,!RD,[AD2,AD1,AD0]] -> [E, RS,RW,D,C,B,A]);


    @const a = 0;
    @repeat 8 {
        [.c., 1, 1, a] -> [.x.,.x.,.x.,.x.,.x.,.x.,.x.,.x.];
        [ 0, .k., 1, a] -> [.x.,.x.,.x.,.x.,.x.,.x.,.x.,.x.] // Schreiben
        [.c., 1, 1, a] -> [.x.,.x.,.x.,.x.,.x.,.x.,.x.,.x.];
        [ 0, 1, .k., a] -> [.x.,.x.,.x.,.x.,.x.,.x.,.x.,.x.] // Lesen
        @const a = a+1;
    }
END

```

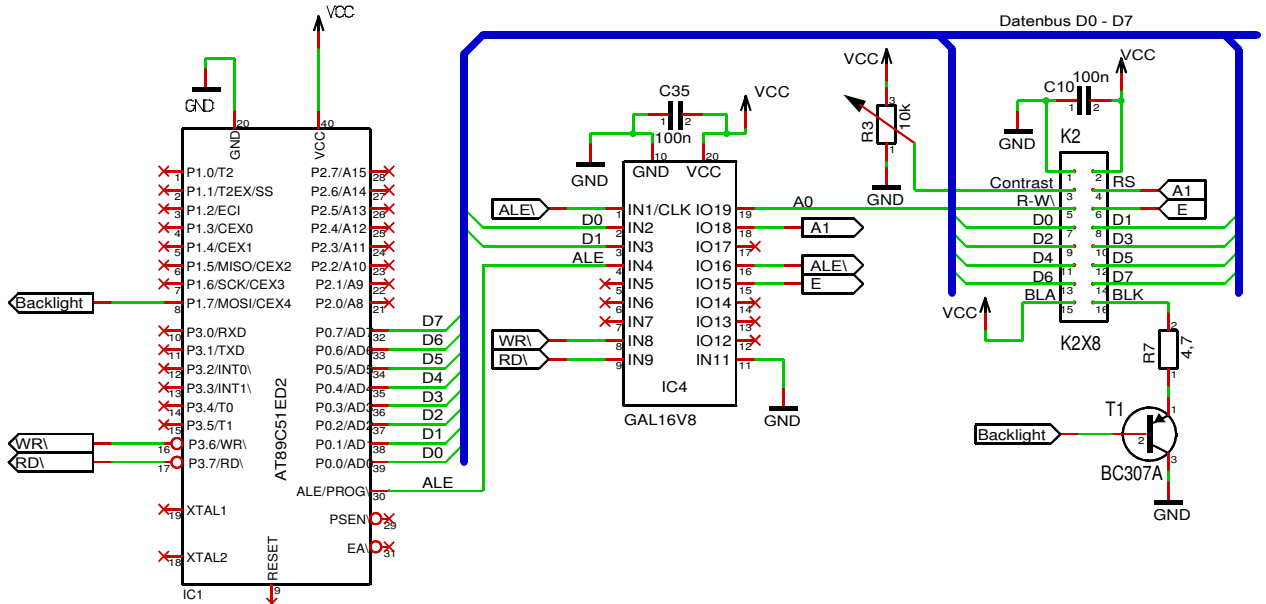
4. Testet die Programmierung mit dem Testvektor mit einer **Functional Simulation!** Der Testvektor erzeugt die folgende Eingangssignale. Dokumentiert die Ausgangssignale:



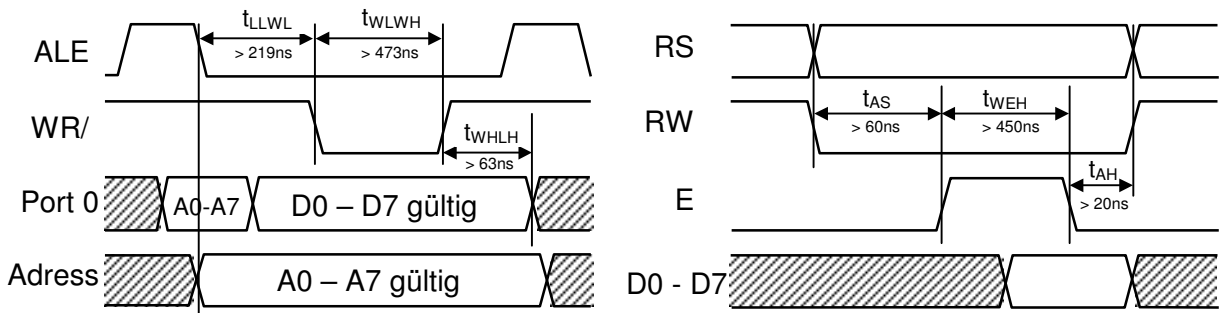
5. Programmiert das JEDEC-File mit dem GALEP IV-Programmiergerät auf den GAL und testet die Programmierung mit dem Programm **LCD_test.hex** auf dem Miniboard!

 Friedrich-Ebert-Schule Esslingen	MIKROCONTROLLER	Name:
	4.6.1	Erzeugung der Steuersignale des LCD

In der abgebildeten Schaltung werden die Steuersignale eines Standard – Textdisplay aus den externen Bussignalen (**AD0, AD1, RD/, WR/**) des Mikrocontrollers gewonnen. Das Adresslatch wird mit einem GAL16V8 realisiert. Es werden nur die Adressen A0 und A1 (= 4 Adressen) benötigt. Da die Adressen bei fallender Flanke von ALE gelatcht werden, die D-Flipflops im GAL jedoch auf steigende Flanke triggern, muss ALE zunächst invertiert werden (ALE/). Das GAL erzeugt die Steuersignale **RS, RW/** und **E** des Display.



Ein Vergleich des Bustiming bei 12MHz Quarztakt und des vom LC-Display geforderte Timing für einen Schreibvorgang zeigt, dass alle Signale über logische Verknüpfung erzeugt werden können:



Auch beim Lesen vom Display werden alle geforderten Zeiten eingehalten. Somit können die Verknüpfungsgleichungen für die Steuersignale des LC-Display formuliert werden:

$$RW = A0$$

$$RS = A1$$

$$E = (\overline{RD/} \wedge \overline{RW}) \vee (\overline{WR/} \wedge \overline{RW})$$

RW = 0: schreiben RW = 1: lesen

RS = 0: Befehl RS = 1: Daten

E = 1: Display-Freigabe (Enable)

Alle Schreib- und Lesebefehle ins DD- und CG-Ram des Displays können nun mit movx -Befehlen erfolgen (R0, R1 als 8 Bit-Adressregister!). Für die vier möglichen Display-Operationen ergeben sich damit folgende Adressen:

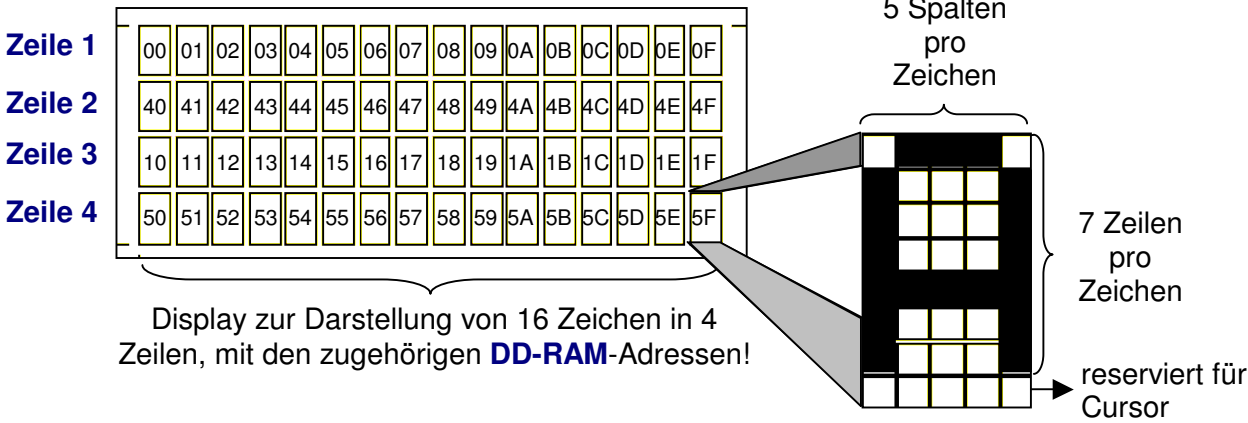
Adressen	hex	Funktion
A 7654 3210	00h	Steuerbefehl schreiben
xxxx xx01	01h	Status lesen (Busy-Flag/Adresse)
xxxx xx10	02h	Daten ins Display schreiben
xxxx xx11	03h	Daten vom Display lesen

Bsp: Schreiben eines Ascii-Zeichens ins Display

```
mov R0,#02h ; Adresse = Daten schreiben
mov a,#'A' ; Ascii-Zeichen
movx @R0,a ; in Display schreiben
```


```
mov R0,#01h ; Busy-Flag lesen
busy: movx a,@R0 ; Wenn Busy dann warten
      jb acc.7,busy; Busy-Flag in Akku-Bit
      .
      .
```

In einem LC-Textdisplay mit HD44780-Controller werden Textzeichen in einer 5x7 dot Punktmatrix dargestellt. Ein ASCII-ähnlicher Zeichensatz ist im **Character-Generator-ROM** (CG-ROM) des Controllers fest abgelegt. Zusätzlich können im 64Byte großen **Character-Generator-RAM** (CG-RAM) bis zu 8 Zeichen (5x7) frei programmiert werden. Zur Darstellung der Zeichen muß der entsprechende Zeichencode auf die gewünschte Adresse im **Display-Data-RAM** (DD-RAM) geschrieben werden.



Zeichenmuster (Pattern) und zugehörige Zeichencodes

Lower 4 bit \ Upper 4 bit	0000 (\$0x)	0010 (\$2x)	0011 (\$3x)	0100 (\$4x)	0101 (\$5x)	0110 (\$6x)	0111 (\$7x)	1010 (\$Ax)	1011 (\$Bx)	1100 (\$Cx)	1101 (\$Dx)	1110 (\$Ex)	1111 (\$Fx)
xxxx0000 (\$x0)	CG RAM (0)	0	@	P	`	P		-	9	3	α	p	
xxxx0001 (\$x1)	(1)	!	1	A	Q	a	q	◻	7	4	ä	q	
xxxx0010 (\$x2)	(2)	"	2	B	R	b	r	┌	ι	×	β	θ	
xxxx0011 (\$x3)	(3)	#	3	C	S	c	s	└	υ	ε	ε	∞	
xxxx0100 (\$x4)	(4)	\$	4	D	T	d	t	√	I	†	μ	Ω	
xxxx0101 (\$x5)	(5)	%	5	E	U	e	u	•	♠	‡	σ	ü	
xxxx0110 (\$x6)	(6)	&	6	F	V	f	v	☞	カ	ニ	ρ	Σ	
xxxx0111 (\$x7)	(7)	'	7	G	W	g	w	ア	キ	ヌ	ラ	g	π
xxxx1000 (\$x8)	CG RAM (0)	<	8	H	X	h	x	イ	ク	ネ	リ	ル	⊗
xxxx1001 (\$x9)	(1))	9	I	Y	i	y	ウ	ケ	ノ	ル	'	y
xxxx1010 (\$xA)	(2)	*	:	J	Z	j	z	エ	コ	ハ	レ	j	〒
xxxx1011 (\$xB)	(3)	+	;	K	[k	[オ	サ	ヒ	ロ	*	斤
xxxx1100 (\$xC)	(4)	,	<	L	¥	l	l	ヤ	シ	フ	ワ	φ	円
xxxx1101 (\$xD)	(5)	-	=	M]	m]	ユ	ズ	ヘ	ン	も	÷
xxxx1110 (\$xE)	(6)	.	>	N	^	n	→	ヨ	セ	ホ	ッ	ñ	
xxxx1111 (\$xF)	(7)	/	?	O	_	o	←	ッ	リ	マ	□	ö	■

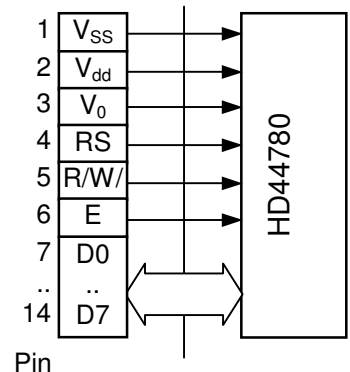
 Friedrich-Ebert-Schule Esslingen	MIKROCONTROLLER	Name:
	4.6.2.2	Aufbau eines LC-Textdisplay

Die Ansteuerung des Display-Controllers durch den Mikrocontroller erfolgt über 8 Datenleitungen D0..D7 und 3 Steuerleitungen. Dies sind die Signale:

- RS** (Register Select): 0: Kommando oder Busy-Flag + Adresszähler
1: Datenregister
- RW/** (read/write): 0: schreiben
1: lesen
- E** (Enable): 1: Schreib- bzw. Lesefreigabe


Für RS und RW/ ergeben sich vier Befehlsgruppen mit unterschiedlicher Wirkung:

RS	RW/	Befehl (Display-Operation)
0	0	Kommando schreiben
0	1	Busy-Flag und Adresszähler lesen
1	0	Daten schreiben
1	1	Daten lesen

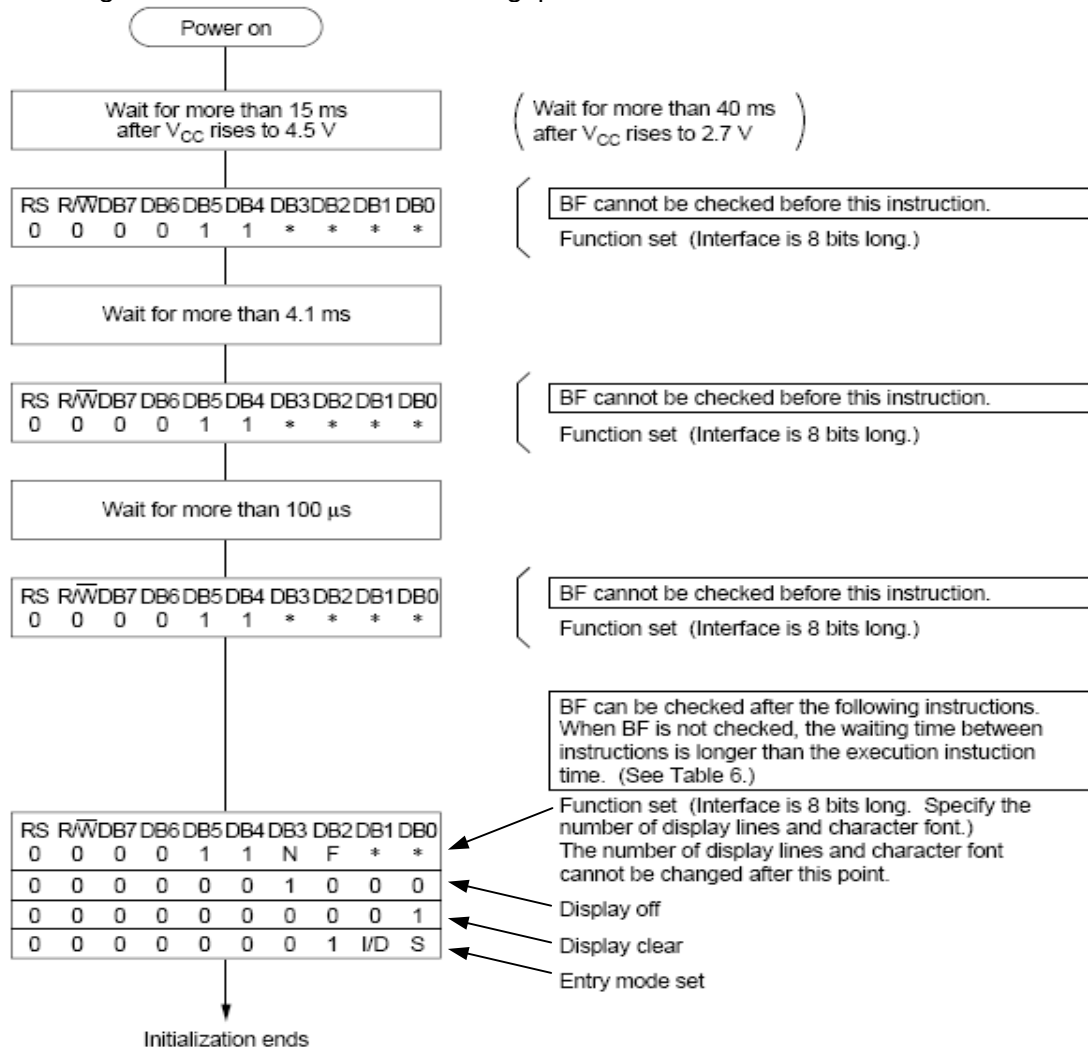


Die Tabelle zeigt alle Kommandos und Befehle des HD44780:

Befehl	Code										Beschreibung
	RS A1	RW A0	D7	D6	D5	D4	D3	D2	D1	D0	
Clear display	0	0	0	0	0	0	0	0	0	1	Löscht das Display und setzt den Cursor auf die Adresse 00h
Cursor home	0	0	0	0	0	0	0	0	1	x	Cursor auf Adresse 00h setzen
Entry mode set	0	0	0	0	0	0	0	1	I/D	S	I/D: 0 = Adress-Zähler dekrementieren 1 = Adress-Zähler inkrementieren S: 0 = Display Shift AUS 1 = Display Shift EIN
Display On/Off control	0	0	0	0	0	0	1	D	C	B	D: 0 = Display AUS 1 = Display EIN C: 0 = Cursor AUS 1 = Cursor EIN B: 0 = Blinken AUS 1 = Blinken EIN
Cursor/display shift	0	0	0	0	0	1	S/C	R/L	x	x	S/C: 0 = Cursorshift 1 = Displayshift R/L: 0 = Links schieben 1 = Rechts schieben
Function set	0	0	0	0	1	DL	N	F	x	x	DL: 0 = 4 Bit Mode 1 = 8 Bit Mode N: 0 = Display Einzeilig 1 = Display Zweizeilig F: 0 = 5x7dots Zeichensatz 1 = 5x10dots Zeichensatz
Set CGRAM address	0	0	0	1	CGRAM Adresse 6 Bit					Nach diesem Kommando werden die nächsten Lese- und Schreiboperationen im Zeichengenerator-RAM durchgeführt.	
Set DDRAM address	0	0	1	DDRAM Adresse 8 Bit					Nach diesem Kommando werden die nächsten Lese- und Schreiboperationen im Display-RAM durchgeführt.		
Read busy-flag and address counter	0	1	BF	Address counter					BF: 0 = Display kann Daten empfangen (ready) 1 = Display ist beschäftigt (busy) Address counter: Z.B. Abfragen der aktuellen Cursorposition		
Write Data to CGRAM or DDRAM	1	0	write data					Schreibt Daten in die aktuelle Adresse und erhöht den Adress-Zähler			
Read Data from CGRAM or DDRAM	1	1	read data					Liest Daten von der aktuellen Adresse und erhöht den Adress-Zähler			

 Friedrich-Ebert-Schule Esslingen	MIKROCONTROLLER	Name:
	4.6.3.1	Initialisierung des LC-Display

Vor dem Anzeigen von Zeichen auf dem Display muss zuerst eine Initialisierung erfolgen. Der PAP zeigt die erforderliche Initialisierungsprozedur nach einem Power-On-Reset :



Im folgenden Listing ist die Initialisierung des Display-Controllers für den Anschluß an den externen Bus des 8051-Controllers gezeigt. Dabei ist **R/W/** mit der Adressleitung A0 verbunden und **RS** mit A1. In der Assembler-Datei wird das Akkubit 7 als **busyflag** und die 4 Display-Adressen als Konstanten deklariert:

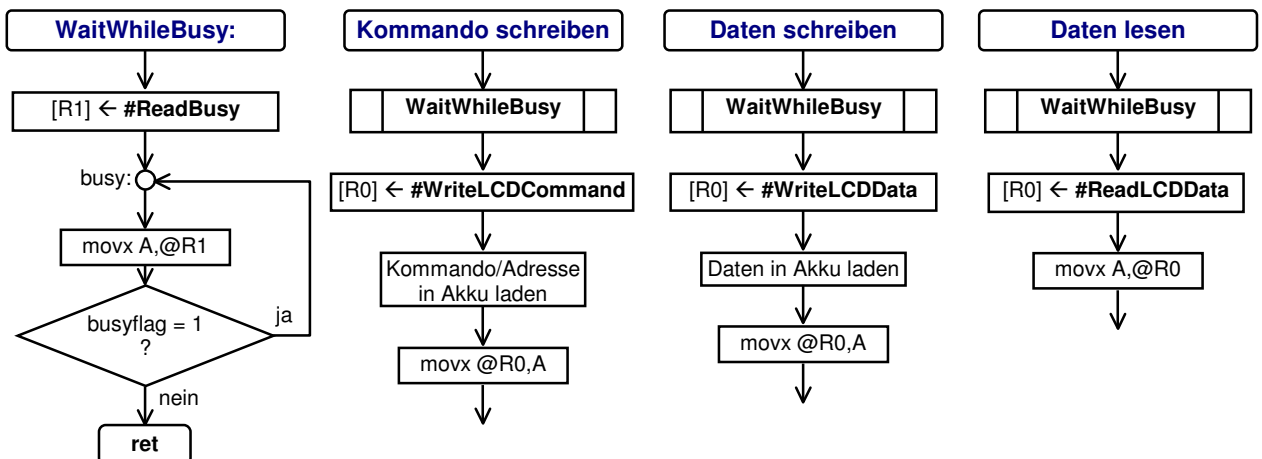
```


busyflag      bit  acc.7           ; Busy-Flag wird in Akkubit 7 zurücklesen!

WriteLCDCommand equ 00h           ; 8 Bit Schreib- und Leseadressen für Display
ReadBusy        equ 01h           ; Lesen mit Movx a,@Rn
WriteLCDData    equ 02h           ; Schreiben mit Movx @Rn,a
ReadLCDData     equ 03h           ; !!! Keinen Datenpointer verwenden !!!
                                   ; (16 Bit Adressierung benötigt P2)

```

Die PAP's zeigen den prinzipiellen Ablauf der vier möglichen Display-Operationen:



 Friedrich-Ebert-Schule Esslingen	MIKROCONTROLLER	Name:
4.6.3.2	Initialisierung des LC-Display	Datum:

Für die Initialisierungsroutine muss gegebenenfalls der externe Bus des Controllers (beim AT89C51ED2) im SFR **auxr** aktiviert werden.

```

;***** Display initialisieren (Anschluß am externen Bus) *****
init_disp:
    mov     auxr,#00000011b           ; Externer Datenbus aktivieren
                                           ; movx erzeugt RD/ und WR/-Impulse

    mov     r7,#48
    lcall   wait                       ; > 15ms warten (48*510µs = 15,3ms)

    mov     r0,#WriteLCDCommand      ; Adresse für "Befehl Schreiben" in R0 laden
    mov     a,#00110000b             ; Function Set: 8 Bit
    movx    @r0,a                     ; LCD aufwecken!

    mov     r7,#9
    lcall   wait                       ; > 4,1ms warten (4,6ms)
    movx    @r0,a                     ; LCD aufwecken!

    mov     r7,#1
    lcall   wait                       ; >100µs warten (510µs)
    movx    @r0,a                     ; LCD aufwecken!

    ; ab hier kann das Busy-Flag abgefragt werden!
    lcall   WaitWhileBusy
    mov     a,#00111000b             ; Function set:
    movx    @r0,a                     ; 8 Bit-Mode / 2 Zeilen / 5x7 Dots

    lcall   WaitWhileBusy
    mov     a,#00001000b             ; Display On/Off Control:
    movx    @r0,a                     ; Display OFF, Cursor OFF, Blink OFF

    lcall   WaitWhileBusy
    mov     a,#00000001b             ; Display Clear
    movx    @r0,a

    lcall   WaitWhileBusy
    mov     a,#00000110b             ; EntryModeSet:
    movx    @r0,a                     ; No shifted Display, Increment

    lcall   WaitWhileBusy
    mov     a,#00001100b             ; Display On/Off Control:
    movx    @r0,a                     ; Display ON, Cursor OFF, Blink OFF

    ret
; Initialisierung beendet

```

Das Unterprogramm **WaitWhileBusy** wird vor jeder neuen Schreib-/Lese-Operation auf das Display aufgerufen. Als Adressregister ist R1 gewählt. Damit muss bei Wiederholung gleicher Operationen nicht jedes mal R0 neu geladen werden.

```

;Warten solange das Display beschäftigt ist!
WaitWhileBusy:
    mov     r1,#ReadBusy             ; Adresse für Busy-Flag lesen in r0

busy:
    movx    a,@r1                     ; read busy-flag & address
    jb     busyflag,busy
    ret

```


Während der Aufweckphase des Displays müssen Wartezeiten programmiert werden, da das Busy-Flag noch nicht abgefragt werden kann.

```

;Unterprogramm: Wartezeit
wait:
    mov     r6,#0ffh
wait_i:
    djnz   r6,wait_i
    djnz   r7,wait

    ret

```

 Friedrich-Ebert-Schule Esslingen	MIKROCONTROLLER	Name:
4.6.4.1	Verwenden der Display-Assemblerbibliothek	Datum:

Zur Nutzung der Bibliotheksfunktionen muss die Datei **lcd_library.a51** ins Projekt eingebunden werden. Weiterhin ist ins Projektverzeichnis (z.B. C:\controller\ed2) die Datei **LCD.inc** zu kopieren. Dort werden globale Symbole zur Displaysteuerung definiert. In jedem Assemblermodul das diese benötigt, muss die Datei inkludiert werden.

LCD.inc

```

;*****
; ** Symboldefinitionen für LC-Display, R.Rahm, 13.12.06
; ** Vor Inklusion dieser Datei muss die Umgebungsvariable LCD16 oder LCD17 gesetzt werden
;*****
; $Set (LCD20) ; Hier die Umgebungsvariable für das verwendete Display
$Set (LCD16) ; setzen! Unbedingt vor Inclusion von LCD.inc

backlight bit p1.7 ; backlight = 0 --> Hintergrundbeleuchtung ein!
busyFlag bit acc.7 ; Busy-Flag wird in Akkubit 7 zurücklesen!

WriteLCDCommand equ 00h ; 8 Bit Schreib- und Leseadressen für Display
ReadBusy equ 01h ; Lesen mit MOVX a,@Rn
WriteLCDData equ 02h ; Schreiben mit MOVX @Rn,a
ReadLCDData equ 03h ; !!! Keinen Datenpointer verwenden !!!
; (16 Bit Adressierung benötigt P2)

; Steuerbefehle
DisplayClear equ 00000001b ; LCD-Befehle
ReturnHome equ 00000010b
CursorOn equ 00001110b
CursorOff equ 00001100b
CursorShiftRight equ 00010100b
CursorShiftLeft equ 00010000b
DisplayShiftRight equ 00011100b
DisplayShiftLeft equ 00011000b

Z1_Start_Address equ 0h ; DDRAM-Adressen für Zeilenanfänge
Z2_Start_Address equ 40h
$if defined (LCD20)
Z3_Start_Address equ 14h ; Abhängig von der Displaygröße
Z4_Start_Address equ 54h
$elseif defined (LCD16)
Z3_Start_Address equ 10h
Z4_Start_Address equ 50h
$endif

```

Die Bibliotheksfunktionen der Datei **lcd_library.a51** sind als Unterprogramme definiert. Dabei ist immer der Akku oder der Datenpointer das Übergaberegister:

Displaysteuerung


<i>init_disp</i>	Initialisierung muss vor jeder Nutzung des Displays durchgeführt werden!
<i>define_char</i>	Definition von eigenen Sonderzeichen
<i>WaitWhileBusy</i>	Das Busy-Flag sollte vor jedem Schreib-/Lesevorgang abgefragt werden!
<i>clear_disp</i>	Gesamtes Display löschen
<i>clear_zeile1</i>	Zeile 1 löschen
<i>clear_zeile2</i>	Zeile 2 löschen
<i>clear_zeile3</i>	Zeile 3 löschen
<i>clear_zeile4</i>	Zeile 4 löschen

Cursorsteuerung

<i>cursor_home</i>	Cursor auf DDRAM Adresse 00h
<i>cursor_rechts</i>	Cursor eine Position nach rechts
<i>cursor_links</i>	Cursor eine Position nach links
<i>set_cursor</i>	Cursor auf beliebige Adresse setzen
<i>cursor_on</i>	Cursor Einschalten
<i>cursor_off</i>	Cursor Ausschalten

Zeichenausgabe

<i>out_char</i>	Ausgabe eines Zeichens auf der aktuellen Cursorposition
<i>out_disp</i>	Ausgabe einer mit 0 terminierten konstanten Zeichenkette
<i>out_var</i>	Ausgabe einer mit 0 terminierten variablen Zeichenkette
<i>text_zeile1</i>	Ausgabe einer 0 terminierten Zeichenkette in Zeile 1
<i>text_zeile2</i>	Ausgabe einer 0 terminierten Zeichenkette in Zeile 2
<i>text_zeile3</i>	Ausgabe einer 0 terminierten Zeichenkette in Zeile 3
<i>text_zeile4</i>	Ausgabe einer 0 terminierten Zeichenkette in Zeile 4

 Friedrich-Ebert-Schule Esslingen	MIKROCONTROLLER	Name:
4.6.4.2	Verwenden der Display-Assemblerbibliothek	Datum:

Vor dem Aufruf der Bibliotheksfunktionen müssen die benötigten Unterprogramme als extern deklariert werden:

```
extern code init_disp
extern code define_char, clear_disp, clear_zeile1, clear_zeile2, clear_zeile3, clear_zeile4
extern code cursor_home, cursor_rechts, cursor_links, set_cursor, cursor_on, cursor_off
extern code out_char, out_disp, out_var, text_zeile1, text_zeile2, text_zeile3, text_zeile4
```

Funktionsaufrufe

- Beispiele für parameterlose Funktionen

```
lcall init_disp
```

```
lcall clear_disp
```

```
lcall cursor_home
```

- Beispiele für Funktionen mit Übergabeparameter im Akku

```
mov a, #'A'
lcall out_char
```

```
mov a, #47h
lcall set_cursor
```

- Beispiel für Funktion mit Übergabeparameter im Datenpointer

```
mov dptr, #text1
lcall text_zeile2
```

...

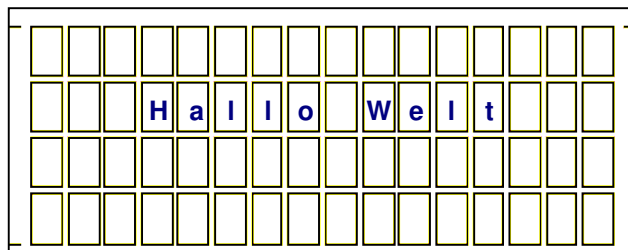
```
text1: db "Hallo", 0
```

- Beispiel für Funktion mit Übergabeparameter in Akku und Datenpointer


```
mov dptr, #zeichen1
mov a, #001000b
lcall define_char
```

Übungen:

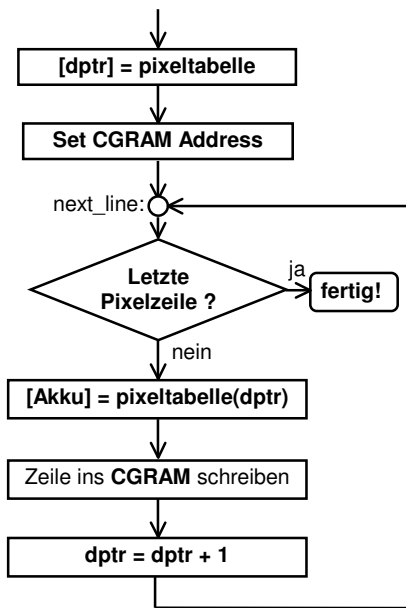
Schreiben Sie ein Assemblerprogramm, welches den Text „**Hallo Welt**“ auf Zeile 2 des LC-Displays ausgibt:



- Verwenden Sie nur die LCD-Funktionen **init_disp** und **Wait_While_Busy**
- Lösen Sie die Aufgabe mit beliebigen Bibliotheksfunktionen

 Friedrich-Ebert-Schule Esslingen	MIKROCONTROLLER	Name:
4.6.5.1	Definieren eigener Sonderzeichen	Datum:

Im Character Generator RAM (CGRAM) können insgesamt 8 eigene Zeichen definiert werden. Ein Zeichen besteht aus 8 Zeilen mit je 5 Pixeln. Jede Pixelzeile hat eine 6 Bit CGRAM-Adresse. Zur Programmierung wird nur die Startadresse über den Befehl **Set CGRAM Address** zum Display-Controller übertragen. Jede Übertragung einer Pixelzeile inkrementiert den Adresszähler danach automatisch. Die Definition eines Zeichens zeigt der folgendem PAP:



Character Codes (DDRAM data)						CGRAM Address						Character Patterns (CGRAM data)										
7	6	5	4	3	2 1 0	5	4	3	2	1	0	7	6	5	4	3	2	1	0			
High			Low			High			Low			High			Low							
										0	0	*	*	*	1	1	1	1	0			
										0	0	1			1	0	0	0	1			
										0	1	0			1	0	0	0	1			
										0	1	1			1	1	1	1	0			
0	0	0	0	*	0 0 0	0	0	0		1	0	0			1	0	1	0	0			
										1	0	1			1	0	0	1	0			
										1	0	1			1	0	0	0	1			
										1	1	0			*	*	*	0	0	0	0	0
										0	0	0			*	*	*	1	0	0	0	1
										0	0	1			*	*	*	0	1	0	1	0
										0	1	0			*	*	*	1	1	1	1	1
										0	1	1			*	*	*	0	0	1	0	0
										1	0	0			*	*	*	1	1	1	1	1
										1	0	1			*	*	*	0	0	1	0	0
										1	1	0			*	*	*	0	0	1	0	0
										0	0	0			*	*	*	0	0	0	0	0
										0	0	1			*	*	*					
0	0	0	0	*	1 1 1	1	1	1		1	0	0										
										1	0	1										
										1	1	0										
										1	1	1										

* don't care

Im folgenden Assemblerprogramm ist die Routine zur Zeichendefinition als Unterprogramm realisiert. Die Startadresse der Zeichencodetabelle wird im Datenpointer übergeben. Im Akku die CGRAM Adresse (6 Bit).

```

mov    dptr,#pixeltabelle
mov    a,#001000b ; CGRAM-Startadresse für Sonderzeichen
lcall  define_char
  
```

Unterprogramm **define_char** zur Definition eigener Sonderzeichen:

```

define_char:
  orl    a,#01000000b ; CGRAM Adresse vorbereiten
  andl   a,#01111111b ; ungültiges Adressbit 7 löschen!
  push  acc
  lcall  WaitWhileBusy
  pop    acc
  mov    r0,#WriteLCDCommand
  movx   @r0,a ; Set CGRAM address

  mov    r0,#WriteLCDData


next_line:
  lcall  WaitWhileBusy ; Busy-Flag und CGRAM-Adresse abfragen
  andl   a,#00000111b ; Lower 3Bit der CGRAM-Adresse ausmaskieren
  cjne   a,#00000111b,n_11 ; Wenn letzte Zeichenzeile erreicht ist,
  ret    ; dann Unterprogramm beenden!

n_11:
  clr    a
  movc   a,@a+dptr ; Zeichencode holen
  movx   @r0,a
  inc    dptr
  sjmp  next_line
  
```

Zeichendefinitionstabelle 8 Pixelzeilen á 5 Pixel:

```

pixeltabelle:
  db    10000b,10000b,10000b,10000b,10000b,10000b,10000b,10000b,10000b
  
```

 Friedrich-Ebert-Schule Esslingen	MIKROCONTROLLER	Name:
4.6.5.2	Definieren eigener Sonderzeichen	Datum:

Zur Darstellung des neu definierten Zeichens auf der aktuellen Cursorposition, muss der DDRAM-Zeichencode auf das Display geschrieben werden.

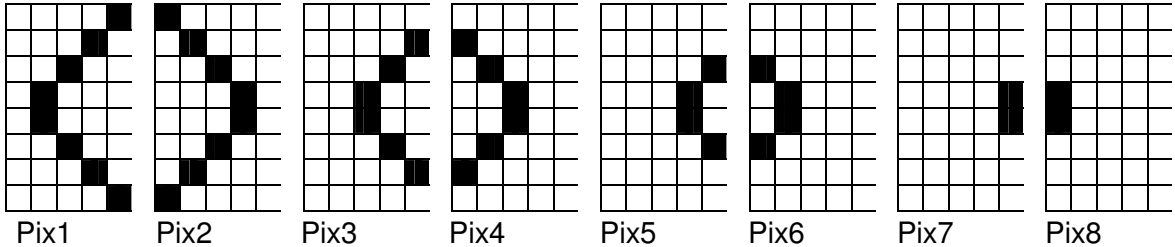
```

mov     a, #01h           ;Zeichencode 01h ausgeben!
lcall  out_char

```

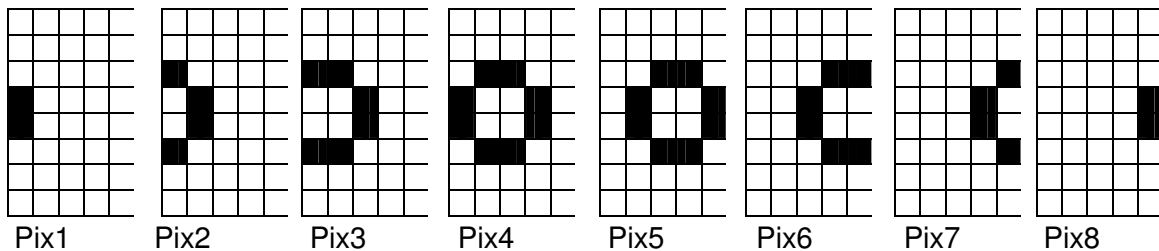
Übung


1. Entwerfen Sie ein Unterprogramm `InitCircles`, das folgende 8 Sonderzeichen definiert:



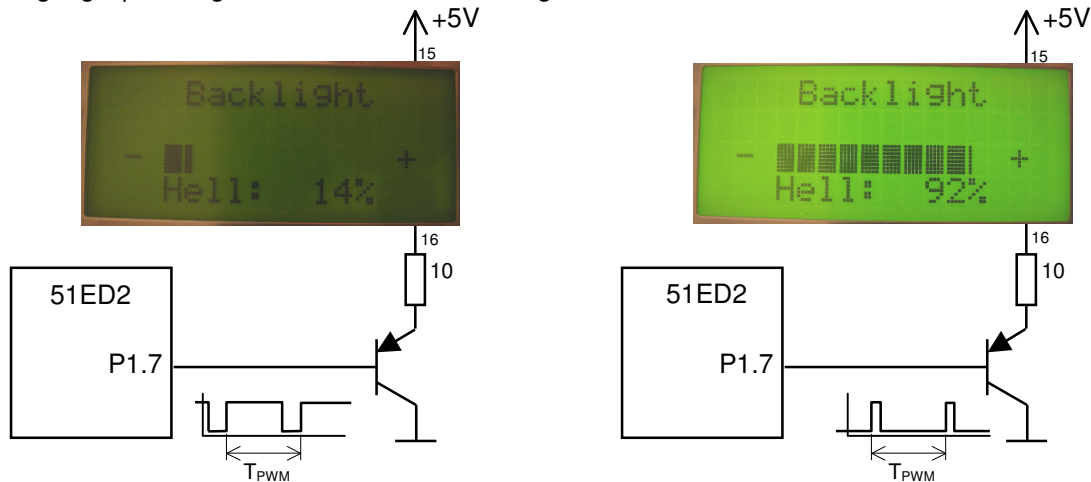
Verwenden Sie das Unterprogramm `define_char`.

2. Die Zeichen sollen jeweils paarweise nacheinander auf den Displayadressen 47h bzw. 48h ausgegeben werden. Nach dem Anzeigen eines Zeichens soll eine kurze Pausenzeit gewartet werden.
3. Der Kreis soll jetzt pulsieren.
4. Mit Hilfe der abgebildeten 8 Zeichen soll ein „Ball“ programmiert werden, der durchs Display rollt!



 Friedrich-Ebert-Schule Esslingen	MIKROCONTROLLER	Name:
	4.6.6.1	LCD-Helligkeitssteuerung mit Bargraphanzeige

Die LED-Hintergrundbeleuchtung eines LC-Display lässt sich durch bloße Spannungsänderung nicht dimmen. Um dennoch die Hintergrundbeleuchtung an die Umgebungshelligkeit anzupassen, wird die Versorgungsspannung durch Pulsweitensteuerung moduliert.



Beim AT89C51ED2 (oder RC2) Controller wird dazu an P1.7 ein PWM-Signal mit dem internen PCA-Modul 4 erzeugt. Zur genauen (und reproduzierbaren) Einstellung der Helligkeit dient eine Bargraphanzeige mit zusätzlicher Prozent-Angabe. Die folgenden Listings und Beschreibungen erläutern die Programmfunktion. Zur Programmierung müssen die Module [LCD_library.A51](#) sowie [LCD_Bargraph.A51](#) ins Projekt (ed2.prj) eingebunden werden. Die Datei LCD.h muss ins Projektverzeichnis kopiert werden.

Backlight_PWM.A51

```

include c51rd2.inc
include lcd.inc           ; ins Projektverzeichnis kopieren

plus_taste      bit    p3.3
minus_taste     bit    p3.2

Prozent         data    7Fh
Delay           data    Prozent-1
Reload         data    Prozent-2

extern code init_disp,text_zeile1,text_zeile4
extern code InitBarGraph,ShowBarGraph,ShowProzent

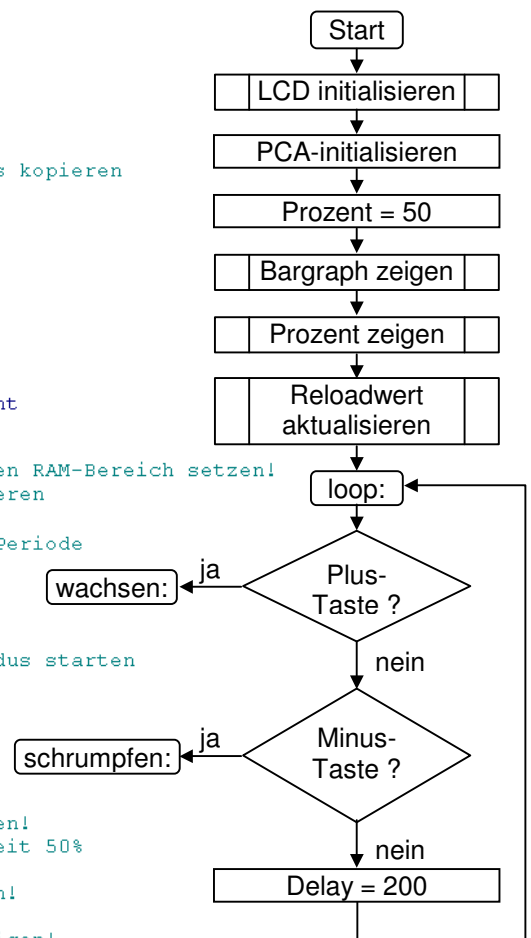
code at 0
start:          mov    sp,#80h           ; Stackpointer auf oberen RAM-Bereich setzen!
               lcall  init_disp         ; LC-Display initialisieren
initPWM:       ; PWM initialisieren
               mov    ccap4L,#7Fh       ; Startwert für 1. PWM-Periode
               mov    ccap4H,#7Fh
               mov    ccapm4,#01000010b
               mov    cmod,#0
               mov    CL,#0
               setb   CR                 ; PCA-Modul 4 im PWM-Modus starten


               mov    dptr,#text1       ; text1 = " Backlight "
               lcall  text_zeile1
               mov    dptr,#text2       ; text2 = " Hell: "
               lcall  text_zeile4

               lcall  InitBarGraph       ; Bargraph initialisieren!
               mov    Prozent,#50        ; Startwert für Helligkeit 50%
               mov    a,Prozent
               lcall  ShowBarGraph        ; Bargraph anzeigen!
               mov    a,Prozent
               lcall  ShowProzent        ; Prozentwert anzeigen!
               lcall  PCA_Reload

loop:          mov    Delay,#200d
               jb    plus_taste,wachsen
               jb    minus_taste,schrumpfen
               mov    Delay,#200d       ; Zeitkonstante fuer Warteschleife
               sjmp  loop

```

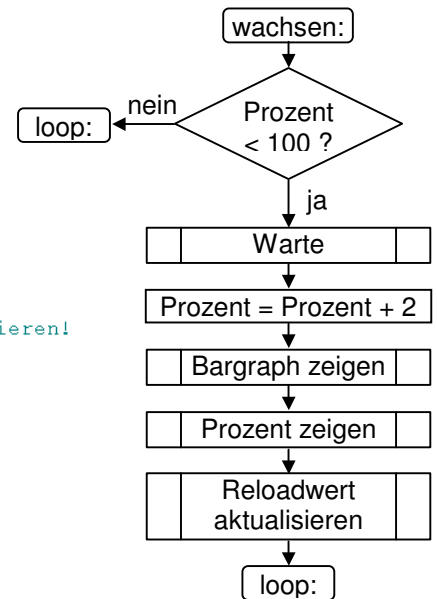


 Friedrich-Ebert-Schule Esslingen	MIKROCONTROLLER	Name:
	4.6.6.2	LCD-Helligkeitssteuerung mit Bargraphanzeige

```

wachsen:
    mov     a,Prozent
    cjne   a,#100,w1      ; Wenn Prozent < 100, dann
    ajmp  loop

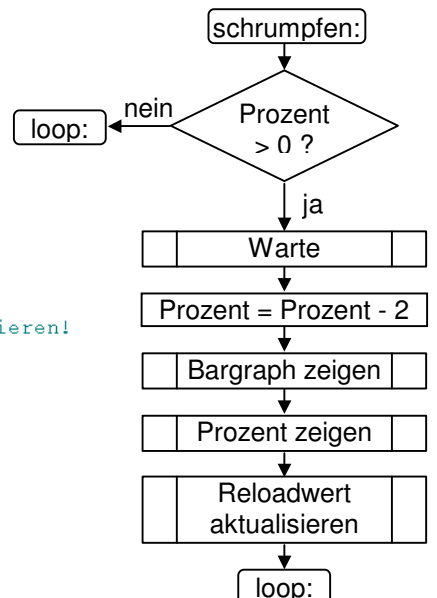
w1:      acall   warte      ;   Wartezeit
         inc    Prozent    ;   Prozent = Prozent + 2
         inc    Prozent
         mov    a,Prozent
         lcall  ShowBarGraph ;   Bargraph anzeigen!
         mov    a,Prozent
         lcall  ShowProzent ;   Prozentwert anzeigen!
         lcall  PCA_Reload  ;   PWM-Reloadwert aktualisieren!
         ajmp  loop
  
```



```

schrumpfen:
    mov    a,Prozent
    jnz   s1      ; Wenn Prozent > 0, dann
    ajmp  loop

s1:      acall   warte      ;   Wartezeit
         dec    Prozent    ;   Prozent = Prozent - 2
         dec    Prozent
         mov    a,Prozent
         lcall  ShowBarGraph ;   BarGraph anzeigen!
         mov    a,Prozent
         lcall  ShowProzent ;   Prozentwert anzeigen!
         lcall  PCA_Reload  ;   PWM-Reloadwert aktualisieren!
         ajmp  loop
  
```



Dynamische Verzögerungszeit

Vor dem Aktualisieren des Bargraphen muss eine kurze Zeit gewartet werden, da der Balken sonst zu schnell durchläuft. Die Verzögerungszeit dient gleichzeitig zur Tastenentprellung. Um die Anzeige bei Tastendruck dynamisch zu verändern, wird der Delay-Wert (= äußerer Schleifenzähler) bei jedem Durchlauf um den Dekrement (#10) erniedrigt, falls er größer als der Endwert (#40) ist. So wird wieder ein zu schnelles Durchlaufen verhindert. Nachdem die Taste losgelassen wurde, muss der Delay wieder auf seinen Startwert (#200) gesetzt werden. Mit Startwert, Dekrement und Endwert kann die Dynamik des Bargraphen an die eigenen Vorlieben angepasst werden. Dabei muss gelten:

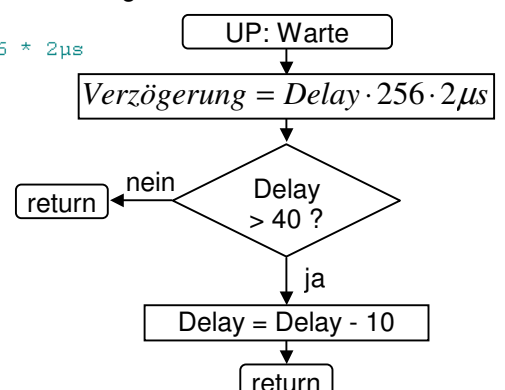
$Endwert = Startwert - n \cdot Dekrement$, wobei n die Anzahl der Durchläufe bei gehaltener Taste ist.


```

warte:   mov    r2,Delay      ; Verzögerung = Delay * 256 * 2µs
d1:      djnz   r3,d1
         djnz   r2,d1
         mov    a,Delay
         cjne   a,#40,d2      ; Wenn Delay > 40,
         ret

d2:      clr    c
         subb   a,#10         ; dann Delay = Delay - 10
         mov    Delay,a

         ret
  
```



 Friedrich-Ebert-Schule Esslingen	MIKROCONTROLLER	Name:
	4.6.6.3	LCD-Helligkeitssteuerung mit Bargraphanzeige

Reloadwert aktualisieren

Das Puls-Pausenverhältnis des PWM-Ausgangs (P1.7) wird durch den 8 Bit-Nachladewert (ccap4h) des PCA-Moduls verändert. Dabei gilt für den Zusammenhang von Nachladewert und Helligkeit (in %):

$$[ccap4h] = 0 \rightarrow 0\% \dots 255 \rightarrow 100\%$$

Die Berechnung des Nachladewertes erfolgt somit durch:

$$Reload = \frac{255 \cdot \text{Prozent}}{100}$$

Diese Berechnung kann mit den 8 Bit Registern des 8051 nicht direkt erfolgen. Daher wird die Formel in 3 einfach zu implementierende Summanden zerlegt:

$$Reload = \frac{200 \cdot \text{Prozent}}{100} + \frac{50 \cdot \text{Prozent}}{100} + \frac{5 \cdot \text{Prozent}}{100}$$

$$Reload = 2 \cdot \text{Prozent} + \frac{\text{Prozent}}{2} + \frac{\text{Prozent}}{20}$$

Da Prozent immer ein gerader Wert ist, entsteht erst im letzten Summanden ein vernachlässigbarer Rundungsfehler:

Bsp.: Prozent = 46

$$Reload = \frac{255 \cdot 46}{100} = 117,3 \quad (\text{genauer Wert})$$

$$Reload = 2 \cdot 46 + \frac{46}{2} + \frac{46}{20} = 92 + 23 + 2R6 = 117 \quad (\text{mit 8051 berechnet})$$

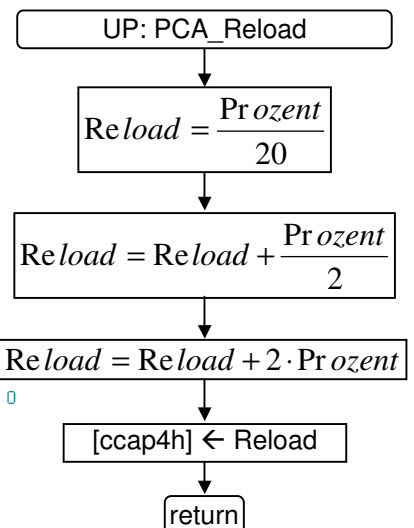
```

PCA_Reload:
    mov     a,Prozent
    mov     b,#20
    div    ab
    mov     Reload,a           ; Reload = Prozent : 20

    mov     a,Prozent
    clr     c
    rr     a
    add    a,Reload
    mov     Reload,a           ; Reload = Reload + (Prozent : 2)

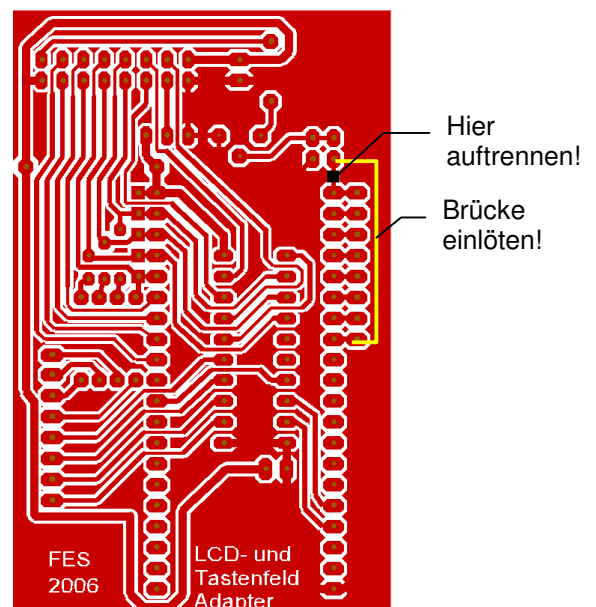
    mov     a,Prozent
    clr     c
    rl     a
    add    a,Reload
    mov     Reload,a           ; Reload = Reload + (Prozent * 2)0
    mov     ccap4h,Reload      ; ccap4h = Reload

    ret
  
```



Änderungen auf der LCD-Adapterplatine

Auf der LCD-Adapterplatine (nur alte Version) wird der PNP-Schalttransistor von P1.0 angesteuert. Als PCA-Ausgänge können aber nur die Ports P1.3 – P1.7 verwendet werden. Um P1.7 (PCA-Modul 4) als PWM-Ausgang zu benutzen, müssen folgende Änderungen an der Platine vorgenommen werden.



 Friedrich-Ebert-Schule Esslingen	MIKROCONTROLLER	Name:
4.6.7	LCD-Displaybefehle mit C	Datum:

Für die Ansteuerung des LC-Displays sind die folgenden Funktionsprototypen in **LCD8.h** definiert:

```

//... Displaysteuerung
extern void InitDisp(void); //Display initialisieren
//Eigene Zeichen definieren
extern void DefineChar(unsigned char *PixelTabelle, unsigned char ZeichenNr);
extern void ClearDisp(void); //Display Löschen
extern void ClearZeile(unsigned char Zeilennummer); //Zeile löschen (1,2,3,4)

//... Cursorsteuerung
extern void CursorHome(void);
extern void CursorRechts(void);
extern void CursorLinks(void);
extern void SetCursor(unsigned char DDRamAddress);
extern void CursorAnfangZeile(unsigned char Zeilennummer);
extern void CursorEin(void);
extern void CursorAus(void);

//... Zeichenausgabe
extern void OutChar(unsigned char Zeichen);
extern void OutDisp(unsigned char *pChar); // Ausgabestring muss /0-terminiert sein
// Ausgabe ab aktueller Cursorposition
extern void TextZeile(unsigned char *pChar, unsigned char Zeilennummer);

```

Die Datei **LCD8.c** muß dem Projekt hinzugefügt werden, oder in die Bibliothek **rc51atms.lib** integriert werden (Anleitung siehe Bubbers). Eine fertige rc51atms.lib befindet sich auf der DVD.

LCD Programmbeispiel:

```

#include <5131.h> // AT89C5131
#include <lcd8.h> // Headerdatei für LC-Display
#include <stdio.h> // wird benötigt für sprintf

code unsigned char Zeichenl[] = { // Tabelle für selbstdefiniertes
    0b00000, // Zeichen 5 x 8 -Matrix
    0b00000,
    0b00100,
    0b01110,
    0b11011,
    0b01110,
    0b00100,
    0b00000
};

void main (void)
{
    unsigned char a;
    unsigned char buffer[16]; // Puffer für Textzeilen

    InitDisp(); // Display initialisieren
    DefineChar(Zeichenl, 0x01); // Erlaubte CGRam-Adressen: 0x01..0x08
    TextZeile("LCDisplay-Test", 1); // konstante Zeichenkette anzeigen

    while (1) // Endlosschleife
    {
        a = P1; // Port 1 einlesen
        sprintf(buffer, "\x01 hex = %X ", a); // Anzeigestring in buffer[]
        // Escape Sequenz \x01 = Adresse 0x01
        TextZeile(buffer, 2); // Ausgabe auf Zeile 2
        sprintf(buffer, "\x01 dez = %u ", a);
        TextZeile(buffer, 3); // Ausgabe auf Zeile 3
        sprintf(buffer, "\x01 char = %c ", a);
        TextZeile(buffer, 4); // Ausgabe auf Zeile 4
    }
}

```

